Axiom AI — Full Project (Frontend + Backend)

This canvas contains the complete copy-paste-ready project files for **Axiom AI** — a Vite + React frontend (Tailwind) and a Node/Express backend that ensembles multiple AI providers. **IMPORTANT:** Never put API keys in the frontend. Put them into the backend `.env`.

Project structure

```
axiom-ai/
├─ frontend/
│  ├─ package.json
│  ├─ index.html
│  ├─ src/
│  │  ├─ main.jsx
│  │  ├─ App.jsx        <- Chat UI
│  │  ├─ index.css
│  │  └─ components/MessageBubble.jsx
│  └─ tailwind.config.js
├─ server/
│  ├─ package.json
│  ├─ index.js         <- Express backend (API aggregator)
│  ├─ aiClients/
│  │  ├─ openai.js
│  │  └─ anthropic.js
│  └─ .env.example
└─ README.md
```

FRONTEND — files to paste

frontend/package.json

```
{
  "name": "axiom-frontend",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "devDependencies": {
    "autoprefixer": "^10.4.14",
    "postcss": "^8.4.24",
    "tailwindcss": "^3.4.7",
    "vite": "^5.0.0"
  }
}
```

frontend/index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Axiom AI</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

frontend/src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './index.css'

createRoot(document.getElementById('root')).render(<App />)
```

frontend/src/index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

html,body,#root { height: 100%; }
```

```
body { font-family: ui-sans-serif, system-ui, -apple-system, 'Segoe UI',
Roboto, 'Helvetica Neue', Arial; }
```

frontend/tailwind.config.js

```
export default {
  content: ['./index.html', './src/**/*.{js,jsx,ts,tsx}'],
  theme: { extend: {} },
  plugins: [],
}
```

frontend/src/components/MessageBubble.jsx

```
import React from 'react'

export default function MessageBubble({ role, text }) {
  let align = 'justify-start'
  let bg = 'bg-gray-100'
  if (role === 'user') { align = 'justify-end'; bg = 'bg-indigo-100 self-end' }
  if (role === 'axiom') { align = 'justify-start'; bg = 'bg-green-50' }
  if (role === 'system') { bg = 'bg-yellow-50' }
  if (role === 'axiom_alt') { bg = 'bg-white border-l-4 border-gray-200' }

  return (
    <div className={`flex ${align}`}>
      <div className={`max-w-[80%] p-3 rounded-lg ${bg} shadow-sm`}>
        <div className="text-sm whitespace-pre-wrap">{text}</div>
      </div>
    </div>
  )
}
```

frontend/src/App.jsx

```
import React, { useState, useEffect, useRef } from 'react'
import MessageBubble from './components/MessageBubble'

export default function AxiomApp() {
  const [message, setMessage] = useState('')
  const [messages, setMessages] = useState([])
  const [loading, setLoading] = useState(false)
  const [connectors, setConnectors] = useState({ openai: '', anthropic: '',
custom: '' })
  const [persona, setPersona] = useState('A powerful, helpful assistant named
Axiom AI. Be concise and bold.')
  const [selectedBrains, setSelectedBrains] = useState(['openai', 'anthropic'])
  const chatRef = useRef(null)

  useEffect(() => { chatRef.current?.scrollIntoView({ behavior: 'smooth' }) },
[messages])

  function addMessage(role, text) { setMessages((m) => [...m, { role, text, id:
Date.now() + Math.random() }]) }

  async function sendMessage() {
    if (!message.trim()) return
    const userText = message.trim()
    addMessage('user', userText)
    setMessage('')
    setLoading(true)

    try {
      const resp = await fetch('/api/axiom/chat', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ prompt: userText, persona, selectedBrains })
      })

      if (!resp.ok) throw new Error('Server error')
      const data = await resp.json()
      if (data.answer) addMessage('axiom', data.answer)
      if (data.candidates && data.candidates.length) {
        addMessage('axiom_notes', '(alternate answers included)')
        data.candidates.forEach((c, i) => addMessage('axiom_alt', `Alt ${i +
1}: ${c}`))
      }
    } catch (err) {
      addMessage('system', 'Error contacting Axiom backend: ' + err.message)
    } finally { setLoading(false) }
  }

  function toggleBrain(name) { setSelectedBrains((s) => (s.includes(name) ?
s.filter((x) => x !== name) : [...s, name])) }

  return (
    <div className="min-h-screen bg-gray-50 text-gray-900">
      <header className="bg-white border-b p-4 flex items-center justify-
between">
        <div className="flex items-center gap-3">
          <div
className="w-10 h-10 rounded-lg bg-gradient-to-br from-indigo-600 to-teal-400
flex items-center justify-center text-white font-bold">AX</div>
```

```
          <div>
            <h1 className="text-lg font-bold">Axiom AI</h1>
            <p className="text-xs text-gray-500">Multi-brain AI fusion — your
own Axiom instance</p>
          </div>
        </div>
        <div className="flex items-center gap-3">
          <button className="px-3 py-1 rounded bg-indigo-600 text-white text-
sm">Sign in</button>
          <button className="px-3 py-1 rounded border text-sm">Deploy</button>
        </div>
      </header>

      <main className="p-6 grid grid-cols-12 gap-6">
        <aside className="col-span-3 bg-white rounded-lg p-4 border h-[70vh]
overflow-auto">
          <h2 className="font-semibold mb-2">Brains & Connectors</h2>
          <p className="text-xs text-gray-500 mb-3">Pick which AI providers
Axiom should call.</p>

          <div className="grid gap-2">
            {[{ id: 'openai', label: 'OpenAI (GPT-like)' }, { id: 'anthropic',
label: 'Anthropic Claude' }, { id: 'mistral', label: 'Mistral / Llama' }, { id:
'stability', label: 'Stability AI (Images)' }, { id: 'eleven', label:
'ElevenLabs (Voice)' }].map((b) => (
              <label key={b.id} className="flex items-center gap-2"><input
checked={selectedBrains.includes(b.id)} onChange={() => toggleBrain(b.id)}
type="checkbox" /><span className="text-sm">{b.label}</span></label>
            ))}
          </div>

          <hr className="my-3" />

          <h3 className="font-medium">API Keys (local demo only)</h3>
          <p className="text-xs text-gray-500 mb-2">Store keys on your backend
— not the frontend for real apps.</p>
          <div className="space-y-2">
            <input value={connectors.openai} onChange={(e) =>
setConnectors({ ...connectors, openai: e.target.value })} placeholder="OpenAI
key (local only)" className="w-full p-2 border rounded" />
            <input value={connectors.anthropic} onChange={(e) =>
setConnectors({ ...connectors, anthropic: e.target.value })}
placeholder="Anthropic key (local only)" className="w-full p-2 border rounded" /
>
            <input value={connectors.custom} onChange={(e) =>
setConnectors({ ...connectors, custom: e.target.value })} placeholder="Custom
API endpoint" className="w-full p-2 border rounded" />
          </div>
          <hr className="my-3" />
          <h3 className="font-medium">Persona</h3>
          <textarea value={persona} onChange={(e) =>
setPersona(e.target.value)} className="w-full p-2 border rounded h-24" />
          <div className="mt-3 flex gap-2">
            <button onClick={() => setPersona('A bold, concise assistant named
Axiom.')} className="px-3 py-1 rounded border text-sm">Quick persona</button>
            <button onClick={() => { setPersona(''); }} className="px-3 py-1
rounded border text-sm">Clear</button>
          </div>
        </aside>

        <section className="col-span-6 bg-white rounded-lg p-4 border h-[70vh]
flex flex-col">
          <div className="flex items-center justify-between mb-3">
            <div>
              <h2 className="font-semibold">Chat</h2>
              <p className="text-xs text-gray-500">Ensembled responses from
selected brains</p>
            </div>
            <div className="text-xs text-gray-500">{loading ? 'Thinking...' :
'Ready'}</div>
          </div>

          <div className="flex-1 overflow-auto p-2 space-y-3">
            {messages.map((m) => (<MessageBubble key={m.id} role={m.role}
text={m.text} />))}
            <div ref={chatRef} />
          </div>

          <div className="mt-3">
            <div className="flex gap-2">
              <input value={message} onChange={(e) =>
setMessage(e.target.value)} onKeyDown={(e) => e.key === 'Enter' &&
sendMessage()} placeholder="Ask Axiom..." className="flex-1 p-3 border
rounded" />
              <button onClick={sendMessage} disabled={loading} className="px-4
py-2 rounded bg-indigo-600 text-white">Send</button>
            </div>
            <div className="mt-2 text-xs text-gray-400">Tip: press Enter to
send. Backend aggregator required to connect multiple AI providers.</div>
          </div>
        </section>

        <aside className="col-span-3 bg-white rounded-lg p-4 border h-[70vh]
overflow-auto">
          <h2 className="font-semibold mb-2">Toolbox</h2>
          <div className="space-y-3">
            <div>
              <h3 className="text-sm font-medium">Templates</h3>
              <p className="text-xs text-gray-500">Prebuilt prompts for
brainstorming, code, and marketing.</p>
              <div className="mt-2 flex flex-col gap-2">
                <button onClick={() => setMessage('Write a startup pitch for a
weather app aimed at skateboarders.')} className="p-2 border rounded text-
sm">Startup pitch</button>
                <button onClick={() => setMessage('Generate a 5-step plan to
scale a freelance brand to $10k/month.')} className="p-2 border rounded text-
sm">Business plan</button>
                <button onClick={() => setMessage('Explain recursion to a 12-
year-old with examples.')} className="p-2 border rounded text-sm">Explain like
I'm 12</button>
              </div>
            </div>

            <div>
              <h3 className="text-sm font-medium">Memory</h3>
              <p className="text-xs text-gray-500">Turn on vector memory to
store the conversation context.</p>
              <div className="flex gap-2 mt-2">
                <button className="p-2 border rounded text-sm">Enable
(Pinecone)</button>
                <button className="p-2 border rounded text-sm">Clear</button>
              </div>
            </div>

            <div>
              <h3 className="text-sm font-medium">Export</h3>
              <p className="text-xs text-gray-500">Export conversation as
markdown or CSV.</p>
              <div className="flex gap-2 mt-2">
                <button className="p-2 border rounded text-sm">Export MD</
button>
                <button className="p-2 border rounded text-sm">Export CSV</
button>
              </div>
            </div>
          </div>
        </aside>
      </main>

      <footer className="p-4 text-center text-xs text-gray-500">Axiom AI •
Frontend demo • Build responsibly</footer>
    </div>
```

```
  )
}
```

BACKEND — files to paste

server/package.json

```
{
  "name": "axiom-server",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "dependencies": {
    "axios": "^1.4.0",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

server/.env.example

```
PORT=3000
OPENAI_API_KEY=sk-REPLACE_ME
ANTHROPIC_API_KEY=claude-REPLACE_ME
```

server/aiClients/openai.js

```
const axios = require('axios')

async function callOpenAI(prompt, opts = {}) {
  const apiKey = process.env.OPENAI_API_KEY
  if (!apiKey) throw new Error('Missing OpenAI key')

  const body = {
    model: opts.model || 'gpt-4o-mini',
    messages: [{ role: 'system', content: opts.persona || '' }, { role: 'user',
content: prompt }],
    max_tokens: 800
  }

  const r = await axios.post('https://api.openai.com/v1/chat/completions',
body, {
    headers: { Authorization: `Bearer ${apiKey}`, 'Content-Type': 'application/
json' }
  })

  return r.data?.choices?.[0]?.message?.content || ''
}

module.exports = { callOpenAI }
```

server/aiClients/anthropic.js

```
const axios = require('axios')

async function callAnthropic(prompt, opts = {}) {
  const apiKey = process.env.ANTHROPIC_API_KEY
  if (!apiKey) throw new Error('Missing Anthropic key')

  const body = {
    model: opts.model || 'claude-v1',
    prompt: `System: ${opts.persona || ''}\nUser: ${prompt}`,
    max_tokens_to_sample: 800
  }

  const r = await axios.post('https://api.anthropic.com/v1/complete', body, {
    headers: { 'x-api-key': apiKey, 'Content-Type': 'application/json' }
  })

  return r.data?.completion || ''
}

module.exports = { callAnthropic }
```

server/index.js

```
require('dotenv').config()
const express = require('express')
const cors = require('cors')
const bodyParser = require('body-parser')
const { callOpenAI } = require('./aiClients/openai')
const { callAnthropic } = require('./aiClients/anthropic')

const app = express()
app.use(cors())
app.use(bodyParser.json())

function ensembleAnswers(answers) {
  const texts = answers.map(a => a.text.trim()).filter(Boolean)
  if (!texts.length) return { answer: 'Sorry, no responses from brains.',
candidates: [] }
  const openai = answers.find(a => a.name === 'openai')
  if (openai && openai.text) return { answer: openai.text, candidates: texts }
  const longest = texts.reduce((a,b) => a.length > b.length ? a : b, texts[0])
  return { answer: longest, candidates: texts }
}

app.post('/api/axiom/chat', async (req, res) => {
  try {
    const { prompt, persona = '', selectedBrains = ['openai'] } = req.body
    const calls = []

    if (selectedBrains.includes('openai')) calls.push(callOpenAI(prompt, {
persona }).then(t => ({ name: 'openai', text: t })).catch(e => ({ name:
'openai', text: '', error: e.message })))
    if (selectedBrains.includes('anthropic')) calls.push(callAnthropic(prompt,
{ persona }).then(t => ({ name: 'anthropic', text: t })).catch(e => ({ name:
'anthropic', text: '', error: e.message })))

    const results = await Promise.all(calls)
    const ensemble = ensembleAnswers(results)

    res.json({ answer: ensemble.answer, candidates: ensemble.candidates, raw:
results })
  } catch (err) {
    console.error(err)
    res.status(500).json({ error: err.message })
  }
})

const PORT = process.env.PORT || 3000
app.listen(PORT, () => console.log('Axiom server listening on', PORT))
```

RUNNING LOCALLY

1. Create folders `frontend` and `server` and paste the files above.
2. In `server`: `npm install` then copy `.env.example` → `.env` and fill API keys.
3. In `frontend`: `npm install`.
4. Run server: `cd server && npm run dev` (or `npm start`).
5. Run frontend: `cd frontend && npm run dev`.
6. Open the Vite URL (usually `http://localhost:5173`). The frontend calls `http://localhost:3000/api/axiom/chat`.

SECURITY + NEXT STEPS

- **Never** commit `.env` with API keys.
- Add rate limiting (`express-rate-limit`), authentication (Auth0 / NextAuth / JWT), billing (Stripe), and vector memory (Pinecone).
- I can add:
- Stripe billing + auth + paid tiers
- Pinecone memory integration
- Docker files and deploy scripts

If you want me to **also** paste Dockerfiles, Stripe + Auth scaffolding, or an automated deploy config for Vercel/Render, say **"add deployment + auth + billing"** and I'll append those files here.

End of canvas.